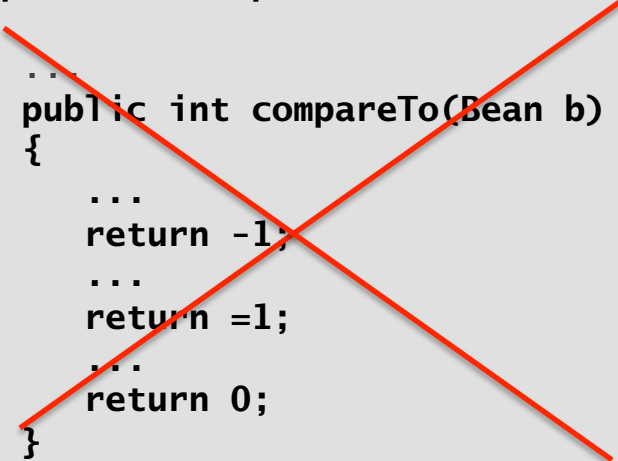

Week 2, Flipped Exercises

Josh Hug
Andy Guna

Comparable interface

```
public class BeanSorter
{
    public static void main(String[] args)
    {
        Bean[] beans = readBeans();
        Insertion.sort(bbeans);
        for (int i = 0; i < files.length; i++)
            StdOut.println(bbeans[i]);
    }
}
```

```
public class Bean
implements Comparable<Bean>
{
    ...
    public int compareTo(Bean b)
    {
        ...
        return -1;
        ...
        return =1;
        ...
        return 0;
    }
}
```



If we omit “compareTo()”, which file will fail to compile?

- A. BeanSorter.java
- B. Bean.java
- C. InsertionSort.java

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

Comparable interface

```
public class BeanSorter
{
    public static void main(String[] args)
    {
        Bean[] beans = readBeans();
        Insertion.sort(beans);
        for (int i = 0; i < files.length; i++)
            StdOut.println(beans[i]);
    }
}
```

```
public class Bean
implements Comparable<Bean>
{
    ...
    public int compareTo(Bean b)
    {
        ...
        return -1;
        ...
        return =1;
        ...
        return 0;
    }
}
```

If we omit “implements comparable”,
which file will fail to compile?

- A. BeanSorter.java
- B. Bean.java
- C. InsertionSort.java

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

Amortized Analysis

- Suppose that you have a class that supports **insert()** and **search()** operations in amortized $4 \lg N$ compares. Which of the following are true?
- I. Starting from an empty data structure, any sequence of N insert and search operations uses at most $4N \lg N$ compares.
- II. Any sequence of N insert and search operations uses at most $4N \lg N$ compares.
- III. Starting from an empty data structure, the expected number of compares to perform N insert and search operations is $4N \lg N$, but there is a (small) probability that it will take $5N \lg N$ compares (or more).

- A. I only.
- B. I and II only.
- C. I and III only.

- D. I, II, and III.
- E. None

Shellsort

- For Shellsort, if we use a $3x+1$ increment sequence, roughly how many elements are considered during the first h sort in an array of size N ?

A. 1
B. 3
C. $N/3$

D. $N^{3/2}$
E. N

Shellsort

- Suppose we have a sorted array of length N . How many compares are required to complete a shellsort with a $3x+1$ increment sequence? Give your answer as an order of growth.
- First few passes:
 - Roughly $N/3$ insertion sorts of size 3 (N compares)
 - ...

A. N
B. $N \log N$

C. $N^{3/2}$
D. N^2

Shellsort

- Suppose we have a sorted array of length N . How many compares are required to complete a shellsort with a $3x+1$ increment sequence? Give your answer as an order of growth.
- First few passes:
 - Roughly $N/3$ insertion sorts of size 3 (N compares).
 - Roughly $N/9$ insertion sorts of size 9 (N compares).
 - ...
 - 1 insertion sort of size N (N compares)
- Total # compares:
 - $N * \log_3(N)$